CSS- notes
21.10.2023 22.43.00

CSS- muistiinpanot

1 Basics of CSS

1.1 Selectors

1. Element Selector:
    - Selects all elements of a particular type.
    - Example: p selects all <p> elements on the page.
2. Class Selector:
    - Selects elements with a specific class attribute.
    - Example: .button selects all elements with class="button".
3. ID Selector:
    - Selects a single element with a specific id attribute.
    - Example: #header selects the element with id="header".
4. Descendant Selector:

- Selects elements that are descendants of a specified element.
- Example: ul li selects all <li> elements that are descendants of a <ul>.

5. Child Selector:
   - Selects elements that are direct children of a specified element.
   - Example: ul > li selects all <li> elements that are direct children of a <ul>.

The difference between children and descencants:

```
<div>
  <ul>
    <li>Direct Child</li> <!-- Selected by ul > li -->
    <li>
      <ul>
        <li>Descendant</li> <!-- Selected by ul li -->
      </ul>
    </li>
  </ul>
</div>
```

6. Adjacent Sibling Selector:
   - Selects an element that is immediately preceded by a specified element.
   - Example: h2 + p selects all <p> elements that directly follow an <h2>.
7. Attribute Selector:
   - Selects elements with a specific attribute and, optionally, a specific attribute value.
   - Example: [type="text"] selects all elements with type="text".
8. Pseudo-classes:
   - Selects elements based on their state or position in the document.
   - Example: a:hover selects all <a> elements when hovered over.
9. Pseudo-elements:
   - Selects a part of an element, such as the first line or the first letter.
   - Example: p::first-line selects the first line of all <p> elements.

1.2 Property and value

”Position” is the property of the element.

“Static” is the value of the property “position”

```
element {
    position: static;
}
```

1.2.1 Miscellanious properties

1.2.2 Miscellanious values

min-content:

min-content sets an element's size to the minimum size necessary to contain its content without overflowing.

It's particularly useful for elements like inline-block elements, which can often become larger than their content requires.

max-content:

max-content sets an element's size to the maximum size it can be without overflowing its containing block or the viewport.

This is helpful when you want an element to be as wide as possible without causing overflow issues.

1.3 Units of measurement

Pixels (px): Pixels are a fixed unit of measurement. One pixel represents one dot on the screen. They are commonly used for setting precise sizes.

Percentage (%): Percentages are relative to the parent element's size. For example, setting the width to `50%` means the element will take up half of its parent's width.

Viewport Width (vw): Viewport width units are relative to the width of the viewport (the browser window). They are useful for creating responsive layouts.

Viewport Height (vh): Similar to `vw`, viewport height units are relative to the height of the viewport.

Ems (em): Em units are relative to the font size of the parent element. If no font size is specified, they are relative to the font size of the element itself.

Root Em (rem): Rem units are similar to `em`, but they are relative to the font size of the root element (usually `<html>`). This makes them more predictable.

Character Width (ch): Ch units are based on the width of the "0" character in the element's font. This can be useful for text-based layouts.

1.4 Positioning

Left, right, top, bottom

`left`: It specifies the distance between the left edge of an element and the left edge of its containing element. You can use it to move an element horizontally to the left or right.

`right`: Similar to `left`, but it specifies the distance between the right edge of an element and the right edge of its containing element. It's used for horizontal positioning as well.

`right`: Similar to `left`, but it specifies the distance between the right edge of an element and the right edge of its containing element. It's used for horizontal positioning as well.

`bottom`: Like `top`, it sets the distance between the bottom edge of an element and the bottom edge of its container. It's used for vertical positioning too, typically moving elements up or down.

Static

This is the default value.

Elements with "position: static" are positioned in the normal document flow. Any top, right, bottom, or left properties have no effect when "position" is set to "static."

Relative

Elements with "position: relative" are positioned relative to their normal position in the document flow.

You can use "top," "right," "bottom," and "left" properties to offset the element from its normal position.

Absolute

Elements with "position: absolute" are positioned relative to the nearest positioned ancestor (an ancestor with a position other than "static") or the initial containing block if there is no positioned ancestor.

The element is taken out of the normal document flow, so it can overlap other elements.

Fixed

Elements with "position: fixed" are positioned relative to the viewport, which means they stay in the same position even when the page is scrolled.

Fixed elements are commonly used for headers, footers, or navigation bars that should remain visible as the user scrolls.

Sticky

Elements with "position: sticky" behave like "position: relative" until they reach a specified scroll position, after which they "stick" to a specific location within their parent container.

You need to set the "top," "right," "bottom," or "left" property along with "position: sticky" to define the position where the element becomes sticky.

1.5 Display values

block:

Elements with display: block; create a block-level box. They typically start on a new line and extend the full width of their parent container. Examples include , <p>, and <h1>.

inline:

Elements with display: inline; generate an inline-level box. They flow within the text and only take up as much width as necessary. Examples include , <a>, and .

inline-block:

Elements with display: inline-block; are similar to inline elements but can have block-level properties and wrap onto a new line when necessary. It's often used for creating inline elements with block-level features.

none:

Elements with display: none; are not displayed on the webpage and take up no space. They are effectively hidden. This property is commonly used for hiding elements via CSS.

table:

Elements with display: table; behave like table elements. They can have table-related properties such as table-row, table-cell, and table-caption. Useful for creating table-like structures without using HTML tables.

list-item:

Elements with display: list-item; generate a list-item box, typically used for creating list items within a list (<ul> or <ol>).

1.6 Layout systems

1.6.1 Flex-box

Flex-direction

Property: flex-direction: row | row-reverse | column | column-reverse;

Description: Determines the main axis along which flex items are placed. You can arrange items horizontally (row or row-reverse) or vertically (column or column-reverse).

Justify-content

Property: justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly;

Description: This property controls the alignment of items along the main axis within the container. It's used to distribute space between or around items.

Align-items

Property: align-items: flex-start | flex-end | center | baseline | stretch;

Description: It defines how flex items are aligned along the cross-axis (perpendicular to the main axis) within the container.

Flex-wrap

Property: flex-wrap: nowrap | wrap | wrap-reverse;

Description: Specifies whether flex items should wrap to the next line when they don't fit within the container. nowrap means no wrapping, wrap wraps items to the next line, and wrap-reverse reverses the wrapping direction.

Align-content

Property: align-content: flex-start | flex-end | center | space-between | space-around | stretch;

Description: This property is used when there is extra space along the cross-axis in a multi-line flex container. It controls how the lines of items are distributed within that space.

Flex-grow

Property: flex-grow: <number>;

Description: Determines how flex items grow within the container when there is extra space available. It's a proportional value that assigns how much of the available space each item should take.

Flex-shrink

Property: flex-shrink: <number>;

Description: Opposite to flex-grow, this property defines how flex items shrink when there is not enough space. It's also a proportional value.

Flex-basis

Property: flex-basis: <length> | auto;

Description: It sets the initial size of a flex item along the main axis before the remaining space is distributed. You can use it to define a fixed size or let it be determined automatically.

Order

Property: order: <integer>;

Description: Allows you to specify the order in which flex items are displayed within the container. Items with lower order values appear first.

1.6.2 Grid

1.6.2.1 Grid templates (defining the grid area and structure)

`grid-template-columns`: Defines the size and structure of grid columns.

`grid-template-rows`: Specifies the size and structure of grid rows.

`grid-template-areas`: Assigns named grid areas to specific grid items.

`grid-template`: A shorthand property to set `grid-template-rows`, `grid-template-columns`, and `grid-template-areas` in one declaration.

1.6.2.1.1 Values to be used with Grid template properties

Fixed Sizes:

You can specify fixed sizes using values like:

`px` (pixels): E.g., `grid-template-columns: 100px 200px;`

`em` or `rem`: E.g., `grid-template-rows: 2em 1em;`

`cm`, `mm`, `in`, `pt`, `pc`, etc.

Relative Sizes:

You can use relative units to create fluid layouts:

`%` (percentage): E.g., `grid-template-columns: 30% 70%;`

Fractional Units: You can use fractions to distribute space proportionally:

`fr` (fraction): E.g., `grid-template-columns: 1fr 2fr 1fr;`

`minmax()` Function: You can use the `minmax(min, max)` function to specify a range of sizes:

E.g., `grid-template-columns: minmax(100px, 1fr) minmax(200px, 2fr);`

`auto`: The `auto` keyword allows a column or row to size itself based on its content:

E.g., `grid-template-rows: 100px auto;`

`repeat()` Function: The `repeat()` function simplifies the declaration of repeated patterns:

E.g., `grid-template-columns: repeat(3, 1fr);`

Named Lines:

You can use named lines to create named areas and reference them in the `grid-template-areas` property:

E.g., `grid-template-areas: "header header" "nav main" "footer footer";`

`none`: In `grid-template-areas`, the `none` value can be used to indicate that a cell should not participate in the grid structure.

Implicit Grid:

Grid template properties can also handle the implicit grid, which is created when the number of items exceeds the defined template. In such cases, the grid can be extended automatically.

1.6.2.2 Implicit Grid properties (grid-auto)

Properties that control the sizing and placement of grid items when they aren't explicitly placed within the grid. These properties are known as "grid-auto" properties and are useful for defining the behavior of items that fall within the implicit grid (i.e., grid items not explicitly placed).

`grid-auto-columns`: Defines the default size of columns in the implicit grid. You can set this property to a specific value, such as `auto`, a fixed size, or a fraction.

`grid-auto-rows`: Specifies the default size of rows in the implicit grid. Like `grid-auto-columns`, you can set this property to various value types.

`grid-auto-flow`: Determines the direction in which grid items are automatically placed within the implicit grid. It can be set to one of the following values:

`row`: Places items in rows, expanding rows as needed.

`column`: Places items in columns, expanding columns as needed.

`dense`: Tries to fill gaps in the grid, ensuring items are placed as efficiently as possible.

1.6.2.3 Aligning Grid items

`grid-column-start` and `grid-column-end`: These properties determine the starting and ending grid lines for items along the horizontal (inline) axis, specifying their column positions.

`grid-row-start` and `grid-row-end`: Similar to `grid-column-start` and `grid-column-end`, but they control the row positions of items along the vertical (block) axis.

`grid-column` and `grid-row`: Shorthand properties for defining both the starting and ending grid lines for columns and rows, respectively.

`grid-area`: Assigns a grid item to a specific grid area, which you define using the `grid-template-areas` property. This property makes it easier to lay out items using named areas.

`justify-self`: Controls the horizontal alignment (justification) of an individual grid item within its cell. Values include `start`, `end`, `center`, and `stretch`.

`align-self`: Similar to `justify-self`, but it controls the vertical alignment (alignment) of an individual grid item within its cell.

`place-self`: A shorthand property that combines `justify-self` and `align-self` to set both horizontal and vertical alignment in one declaration.

`grid-column-gap` and `grid-row-gap`: These properties set the size of gaps (spacing) between columns and rows within the grid.

`grid-gap`: A shorthand property to set both `grid-column-gap` and `grid-row-gap` in one declaration.

`z-index`: Although not exclusive to CSS Grid, you can use the `z-index` property to control the stacking order of grid items in the grid, which is particularly useful when dealing with grid items that overlap.

1.7 Pseudo selectors

Define the special state of an HTML element. They allow you to select elements based on criteria that cannot be targeted using simple selectors alone, such as their position in the document, user interactions, or specific attributes.

:root:

Represents the highest-level parent element in the Document Object Model (DOM) tree, which is typically the <html> element.

It is often used to define global CSS variables (custom properties) that can be accessed and reused throughout your CSS stylesheet.

```
:root {
  --main-color: #007bff; /* Define a variable named --main-color and set its value to blue */
}

body {
  background-color: var(--main-color); /* Use the variable --main-color as the background color */
}
```

:hover:

Selects an element when the mouse pointer is placed over it. Useful for creating hover effects.

:active:

Selects an element when it's being activated, such as when a button is clicked.

:focus:

Selects an element when it has keyboard focus, such as when a user clicks on an input field.

:first-child:

Selects the first child element of a parent.

:last-child:

Selects the last child element of a parent.

:nth-child(n):

Selects elements based on their position within their parent. You can use specific formulas like even, odd, or numeric values.

:not(selector):

Selects elements that do not match a given selector.

:nth-of-type(n):

Similar to :nth-child, but selects elements based on their type within their parent.

:checked:

Selects input elements (like checkboxes or radio buttons) that are checked.

:disabled and :enabled:

Selects elements that are disabled or enabled, respectively.

1.8 Pseudo elements

Cannot directly be accessed with JavaScript!

Allow you to style a specific part of an HTML element. They are denoted by double colons (::) and are used to target elements' content that can't be selected using regular CSS selectors alone.

::before:

Used to insert content before the content of an element.

Often used for adding decorative elements or icons before an element's content.

```
p::before {
    content: "→ ";
    color: blue;
}
```

::after:

Used to insert content after the content of an element.

Similar to ::before but adds content after the element's content.

```
button::after {
    content: " (click me)";
    font-weight: bold;
}
```

::first-line:

Selects the first line of text within an element.

Allows you to apply styles specifically to the first line of text.

::first-letter:

Selects the first letter of text within an element.

Useful for creating drop caps or applying unique styles to the first letter.

::selection:

Targets the portion of text that has been selected by the user.

You can style the selected text, such as changing its background color or text color.

::placeholder:

Used to style the placeholder text within form input fields.

Allows you to apply styles to the text that appears before the user enters any input.

::marker:

Selects the marker box of a list item (e.g., bullet points or numbers).

Useful for customizing list markers.

::backdrop:

Targets the backdrop layer of a modal or dialog element.

Allows you to style the background behind the modal.

1.9 Media queries

1.10 Animations and transformations

1.10.1 Common animations properties and their values

Transform:

Used to apply 2D and 3D transformations to elements. It allows you to modify an element's size, position, and rotation without affecting the layout or flow of the document.

Example:

```
transform: scale(1.5);
```

Transition:

Animate changes to an element over a specified duration. It specifies the properties that should be transitioned, the duration of the transition, the timing function, and a delay if needed.

Property: Specifies the CSS property to transition. This is usually the name of the property you want to animate, such as `background-color`, `transform`, or `width`.

Duration: Defines how long the transition should take to complete. You can specify the time in seconds (e.g., `0.5s`) or milliseconds (e.g., `500ms`).

Timing Function: Describes the speed curve of the transition. Common timing functions include:

`ease`: Starts slow, speeds up, and then slows down (default).

`linear`: Progresses at a constant speed.

`ease-in`: Starts slow and accelerates.

`ease-out`: Starts fast and decelerates.

`ease-in-out`: Combines elements of both `ease-in` and `ease-out`.

Delay (optional): Specifies a delay before the transition begins. You can use seconds (e.g., `1s`) or milliseconds (e.g., `100ms`) for the delay.

Example:

```
/* Basic syntax */
element {
  transition: property duration timing-function delay;
}
```

1.11 Functions in CSS

url() = specify the location of a resource, such as an image, font or another exterfile or a website withing a CSS property value.